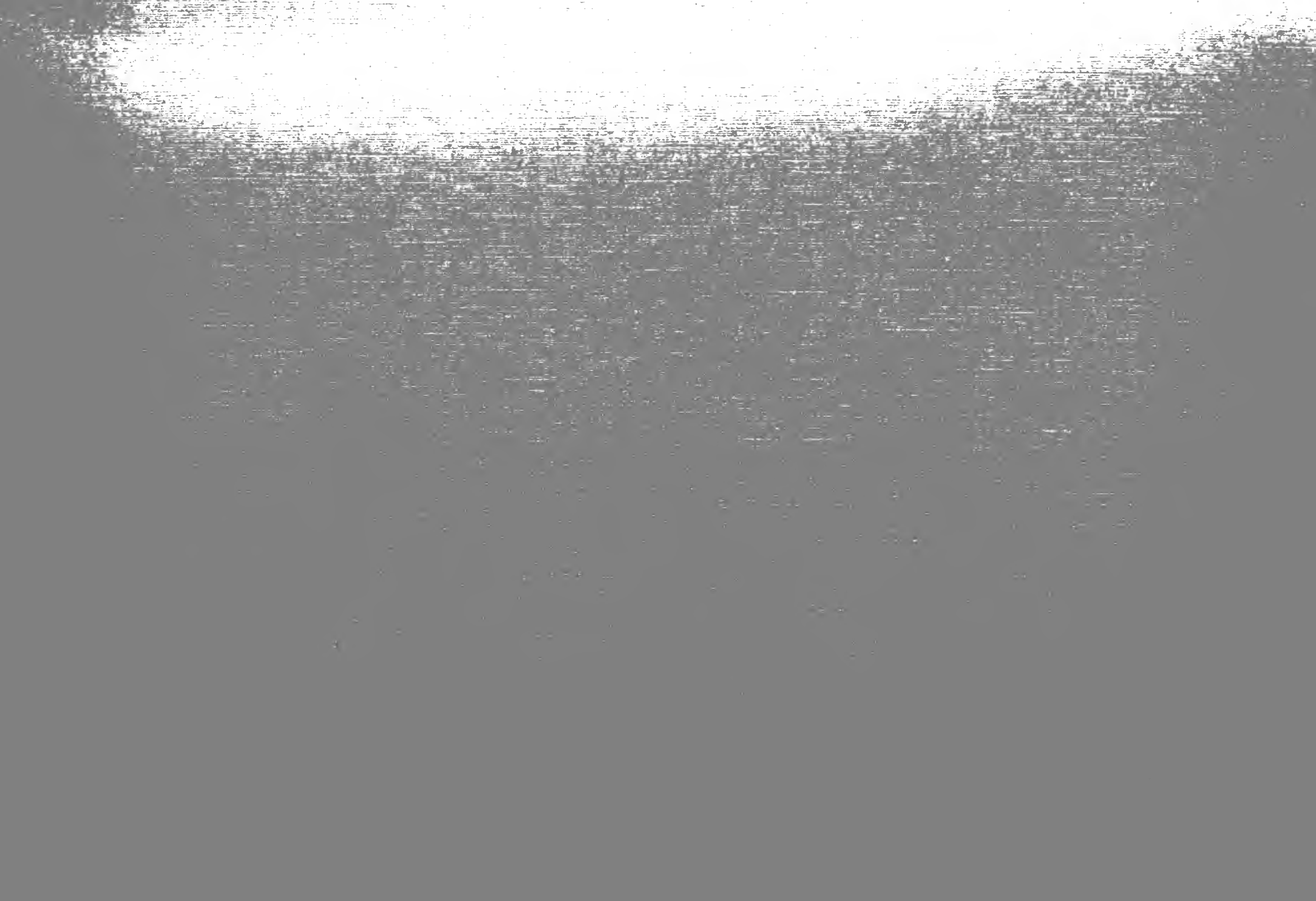
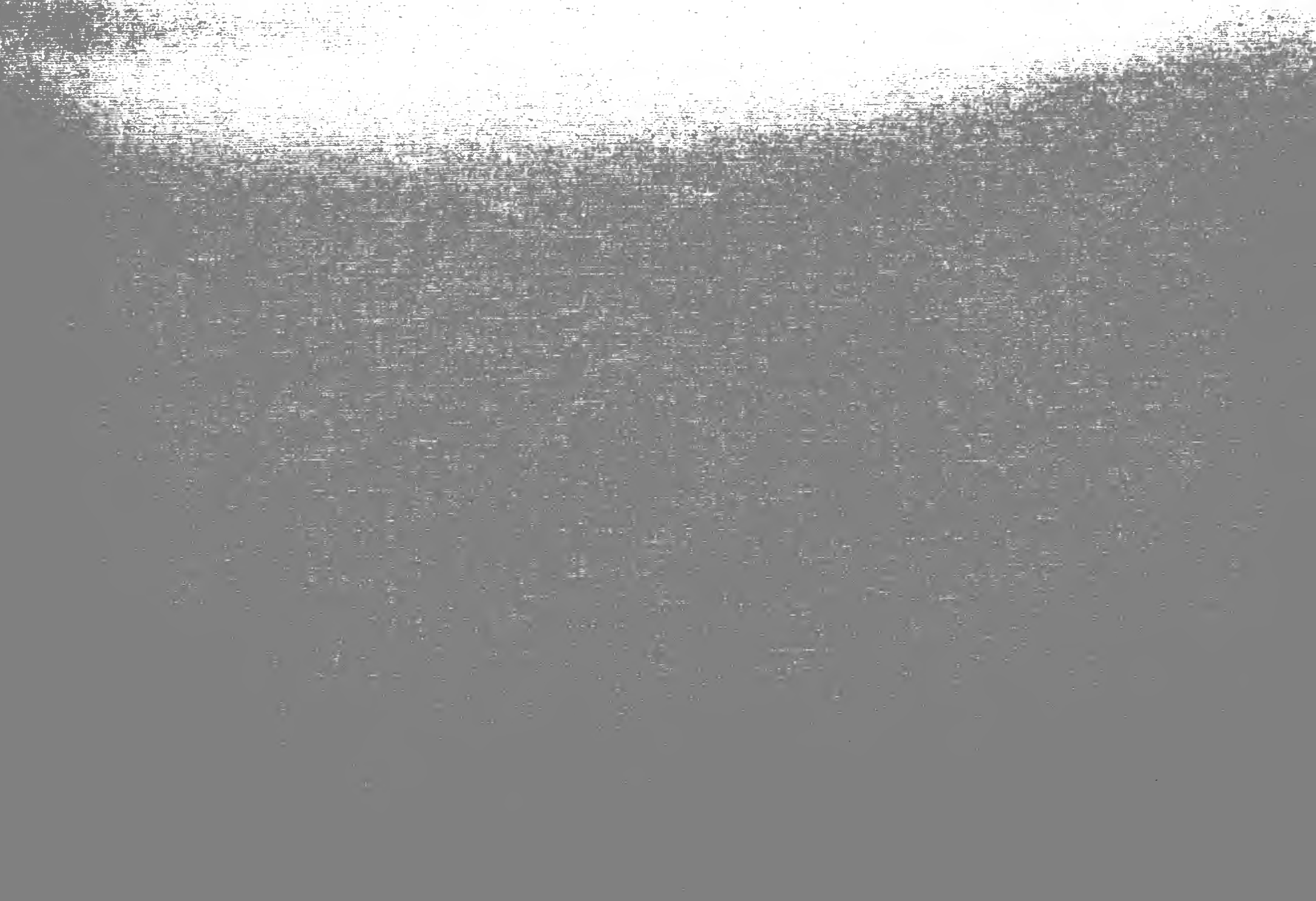


BEBR
FACULTY WORKING PAPER NO. 1351

Knowledge-Based Scheduling in Flexible Manufacturing Systems

Michael J. Shaw

College of Commerce and Business Administration
Bureau of Economic and Business Research
University of Illinois, Urbana-Champaign



BEBR

FACULTY WORKING PAPER NO. 1351

College of Commerce and Business Administration

University of Illinois at Urbana-Champaign

April 1987

Knowledge-Based Scheduling in Flexible Manufacturing Systems

Michael J. Shaw. Assistant Professor
Department of Business Administration

Abstract

This paper presents a knowledge-based scheduling approach based on the problem-solving techniques developed in artificial intelligence. It models the scheduling process by state-space transitions; the job routing is obtained through selecting a sequence of scheduling operators guided by heuristics. Keeping track of the manufacturing system by a symbolic world model, this approach is adaptive to such environmental changes as new job arrivals and machine breakdowns, suitable for making real-time scheduling decisions. A scheduling procedure based on the A* algorithm is described, in which various types of scheduling heuristics can be taken into account. The paper concludes with a computation study reporting the performance resulting from these heuristics.

I. Introduction

The recent advant of manufacturing and computer technologies has accelerated the realization of computer integrated manufacturing where manufacturing processes are fully automated with computer-assisted decision-making capability. Flexible manufacturing systems (FMSs), the type of manufacturing systems that can produce a variety of items efficiently, have become increasingly important in the effort to improve the productivity of batch manufacturing. An FMS consists of a number of computer-controlled machines or workstations, an automated material handling system which transports workpieces between any pair of machines, and a number of supervisory computers networked together for on-line control. By integrating the versatile numerical control (NC) machines with the real-time decision-making capability of the supervisory computers, an FMS is able to reduce the flow-time and the set-up time significantly and, thus, achieve much needed efficiency in manufacturing a variety of items simultaneously.

The production environment in an FMS is a highly dynamic one. The machines are versatile and capable of performing a variety of operations with different set-ups. The workpieces can be transported among machines by mobile robots, automated guided vehicles, or conveyors. Due to the flexibility of the system, a given operation usually can be performed by a number of machines; the decision on assigning a job to a machine depends to a great extent on the situation at the participation moment, that is, the decision is state-dependent. In addition, sometimes there may be needs for cancelling or reassigning machines or other

resources because of unexpected breakdowns. Thus, the scheduling decision is a complex one which needs to be adaptive to environment changes (Cutkosky et al. (1983), Ranky (1983), Merchang (1983)).

This paper takes a knowledge-based approach to the FMS scheduling problem drastically different with the conventional scheduling methods. It models the scheduling process by state-space transitions; the job routing is obtained through selecting a sequence of scheduling operators guided by heuristics. Keeping track of the manufacturing environment by a symbolic world model in the knowledge-based scheduler, this approach is adaptive to such environmental changes as new job arrivals and machine breakdowns, thus suitable for dynamic scheduling/rescheduling.

The traditional scheduling approaches usually employ one of the three classes of techniques: network analysis, combinational methods, and dispatching heuristic procedures. However, network analysis is usually based on a predetermined network and, thus, is inadequate for dynamic scheduling where the precedence network is constantly changing. Combinatorial method, also restricted to static scheduling, suffers from the complexity problem (i.e., combinatorial explosion) which is difficult to overcome for large problems. The heuristic scheduling procedures use relatively simple knowledge representation and are restricted to limited intelligence--they use rigid heuristic which cannot adapt to environmental changes (Grant (1986), Brund et al. (1986), and Kusiak (1986b)). In contrast, the approach presented in this paper can dynamically generate schedules represented by partially ordered networks; it can incorporate heuristic knowledge to expedite the scheduling process; lastly, it is equipped with a structured representation scheme

based on the same inference organization used as an information processing model for human problem solving (Hayes-Roth and Waterman (1978), Newell and Simon (1972)).

The remainder of this paper is organized as follows. Section II gives an overview of the knowledge-based scheduler and the information organization of the knowledge-based system. Section III describes the heuristic searching procedure used by the knowledge-based scheduler for constructing schedules. Section IV explores a decomposition approach to scheduling with reduced complexity. Section V shows the implementation results and a computation study on the impact of various types of heuristic knowledge on the performance of the knowledge-based scheduler.

II. Background

II.1. The FMS Scheduling Problem

The scheduling of jobs in an FMS has been addressed by a number of researchers within a hierarchical framework, where the scheduling decision consists of three levels (Buzacott (1982), Hitz (1979), Nof et al. (1979), Stecke (1983), Gershwin et al. (1985), Buzacott and Yao (1986)):

Level 1: Prerelease planning - Deciding the job mix to be manufactured and the constraints on operation sequence.

Level 2: Job entry control - Determining the sequence and timing of release of job.

Level 3: Operation and flow control - Ensuring the best routings for the jobs based on the availability of machines and other resources.

This paper focuses on the scheduling aspect at the operation and flow control level of the decision hierarchy. That is, if the production mix of jobs is specified and the job entry sequence is known, to determine the machine assignments and manufacturing paths for the jobs dispatched into the system. Such a scheduling decision can be made by enumerating the manufacturing paths in advance, or by establishing a set of decision rules which determine machine assignment in real time as the workpiece makes its way through the system (Buzacott, 1982). In the FMS environment, where the machines are flexible but subject to failures, the ability to schedule jobs in real time and to reschedule jobs dynamically is desirable. Hitz (1979) used a periodic scheduling algorithm to evaluate schedules, but the approach required that the job routings be determined in advance. Kimemia and Gershwin (1985) developed optimization techniques for solving the flow control problem by mathematical programming. Park and Shaw (1986) incorporated sequencing heuristics to integrate the job entry and flow control level in FMS scheduling. Fox (1983) used a constraint-directed approach which integrated the scheduling problems on the three levels. Buzacott and Yao (1986) provide a comprehensive survey on the analytical techniques that have been developed and used in all three levels of the decision hierarchy.

The inference procedure of the knowledge-based scheduler is similar to that of an automatic planning system in the AI literature (Fikes and Nilsson (1971), Sacerdoti (1977), Vere (1983), Wilkins (1984)). Conceptually, an automatic planning system develops a course of action, or a "plan," for the agents to reach the goals desired; the plan will then

be used to guide the execution of planned activities. In flexible manufacturing, the agents--which may be robots, computerized machine-centers, or the host computer of a manufacturing cell--can carry out a variety of operations, including various types of machining, workpiece routing, loading, and unloading operations.

Furthermore, the knowledge-based scheduler we developed is organized as a hierarchical system consisting of three levels: strategic level, planning level, and operational level (Figure II.1). Because of the scheduling characteristics of the flexible manufacturing cell, the strategic level can choose four different planning modes: static planning, dynamic planning, plan-revision, and simulation. The inference engine can execute either forward- or backward-chaining to construct schedules. By this design, the knowledge-based scheduler can perform the following types of scheduling functions:

(1) Adaptive scheduling: Schedules are generated by goal-driven procedures; the scheduler can perform dynamic scheduling to adapt to changes in the FMS environment.

(2) Planning: State-space inference and heuristic search are used to derive production processes.

(3) Optimizing: Simulation and selection of plans from alternates are done by evaluating performance criteria and heuristics.

(4) Learning: Recognition, refinement, encoding, and integration of processes are performed to enhance the scheduling performance.

This paper will focus on the first three aspects of knowledge-based scheduling.

Insert Figure II.1 Here

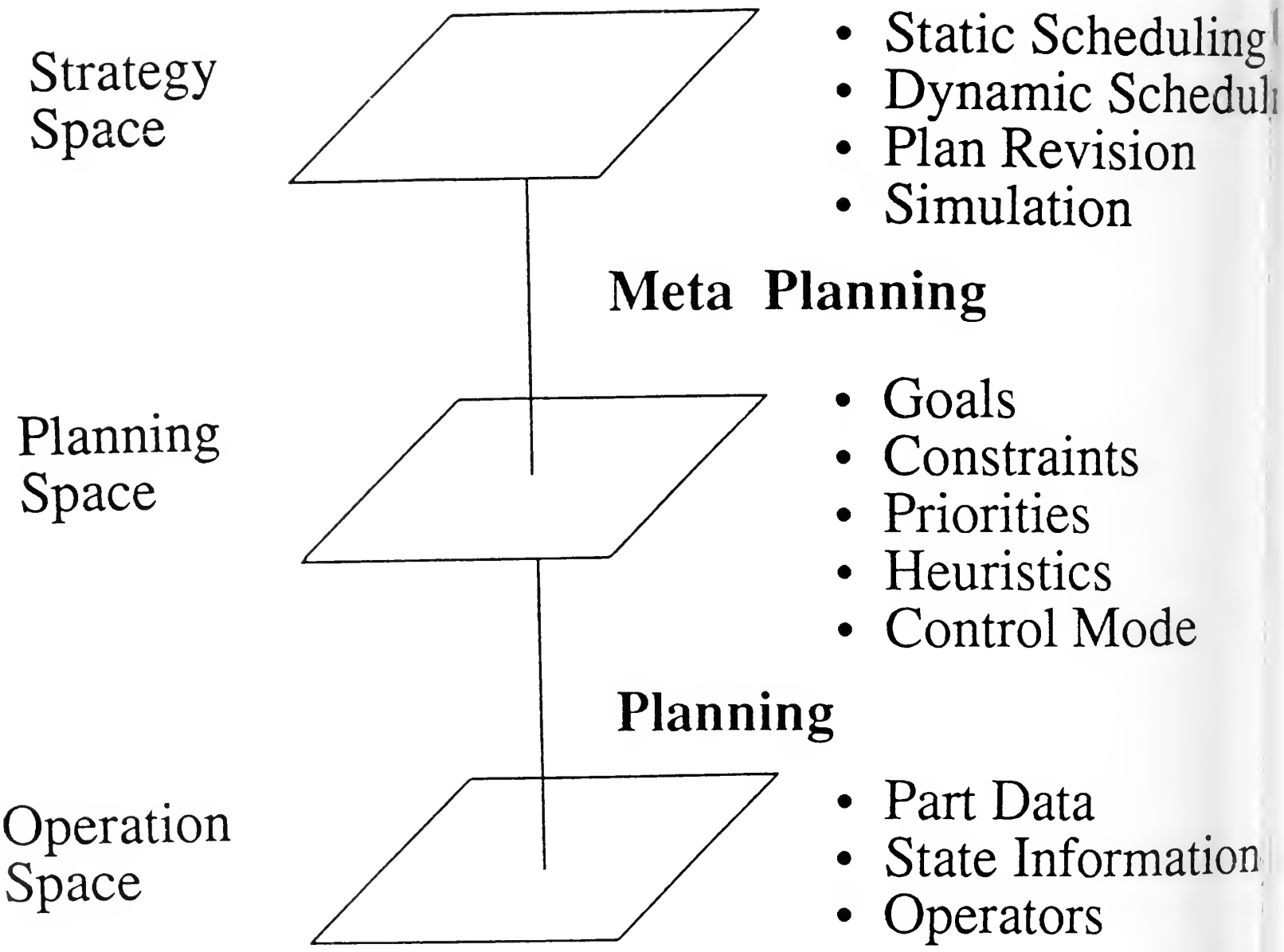


Figure II.1 The Hierarchical Structure of the Knowledge-Based System

II.2. The Organization of the Scheduling System

A general knowledge-based system consists of three components: a database, a knowledge base, and an inference engine. The database stores declarative knowledge about the goals, the current situation of the world, and the semifinished plan. The knowledge-base stores the domain-specific and procedural knowledge, often represented by production rules or operators. Finally, the inference engine stores control knowledge indicating how to select operators and when to apply them.

Because scheduling involves exploration of alternative sequences of actions, a symbolic model of the real world, the "world model," represents the manufacturing environment in the scheduling process. For any given scheduling problem, the initial condition and the stated goal condition are both treated as instances in the world model. The function of the knowledge-based scheduling system, then, is to construct a course of action that transforms one state of the world model, which contains an initial condition, to a state which satisfies the goal condition. Thus the organization of the knowledge-based scheduling system consists of the following three components:

(1) The database. Sometimes referred to as the blackboard, the database stores the world model and partially constructed schedules. The world model consists of symbolic descriptions of the real world; this world model is represented by the collection of first-order predicates in the database and is used to contain state-descriptions of the FMS environment in the scheduling process.

(2) The knowledge base. The principal content of the knowledge-base describes the transformational effects of actions that map states

to other states. Such transformations are usually modeled by operators similar to the STRIPS operators defined in Nilsson [1980]. In such an action model, each operator can be specified as follows:

< Action - name > : < list-of-arguments >

< Precondition > : < list-of-precondition-literals >

< Add list > : < list-of-add-list-literals >

< Delete list > : < list-of-delete-list-literals >

< Resource > : < resource-name >

< Duration > : < length-of-duration >

In addition to the standard STRIPS formalism--which specifies an action by the add list, delete list, and preconditions--we have also included two more descriptions for each action--the "resource" used during the action, and the "duration" of the action. Some sample operators are shown in the Appendix. The knowledge-base also contains manufacturing knowledge in the forms of heuristics and constraints.

(3) The inference engine, which directs the schedule-generation process. It selects operators to achieve the goal state from a given initial state. The inference engine employs a heuristic searching procedure which will be discussed in more details in Section III.

III. Schedule Generation by Heuristic Searching

The scheduling process can be viewed as finding the solution path in a search tree, such as the one shown in Figure III.1, carried out by either forward- or backward-chaining. In forward-chaining, the root of the tree is the initial state, and instances of operators define the

branches. Searching for solutions is accomplished by a "generate-and-test" process (Newell and Simon, 1972): at a given node, corresponding to a state in the world model, new nodes are generated by applying operators (rules). This process continues until the goal state is generated. The backward chaining process operates in reverse direction, i.e., starting with the goal and reaching for the initial conditions. The key issue, then, is the selection of the most appropriate operator to apply at a given state, represented as a node in the search tree. The heuristic knowledge is important for expediting the searching of operators. The schedule-generation process can be achieved by the following heuristic search procedure, called the A^* algorithm in the literature (Nilsson, 1980; Peal, 1983):

Algorithm 3.1 (Operator-Search)

Input:

S_0: the initial state
S_G: the goal state
OP: the set of operators

Output:

ψ: a linked list of instantiated operators.

Begin

1) Place S_0 on a list called OPEN; create a list called CLOSED that is initially empty.

2) LOOP: If OPEN is empty, exit with failure.

3) Remove from OPEN and place on CLOSED a node n for which f(·) is minimum.

4) If n matches S_G, exit successfully with the plan constructed by tracing the solution path from n to S_0. Concatenate the operators in the order of their application to form the plan ψ. (Pointers are established in Step (5).)

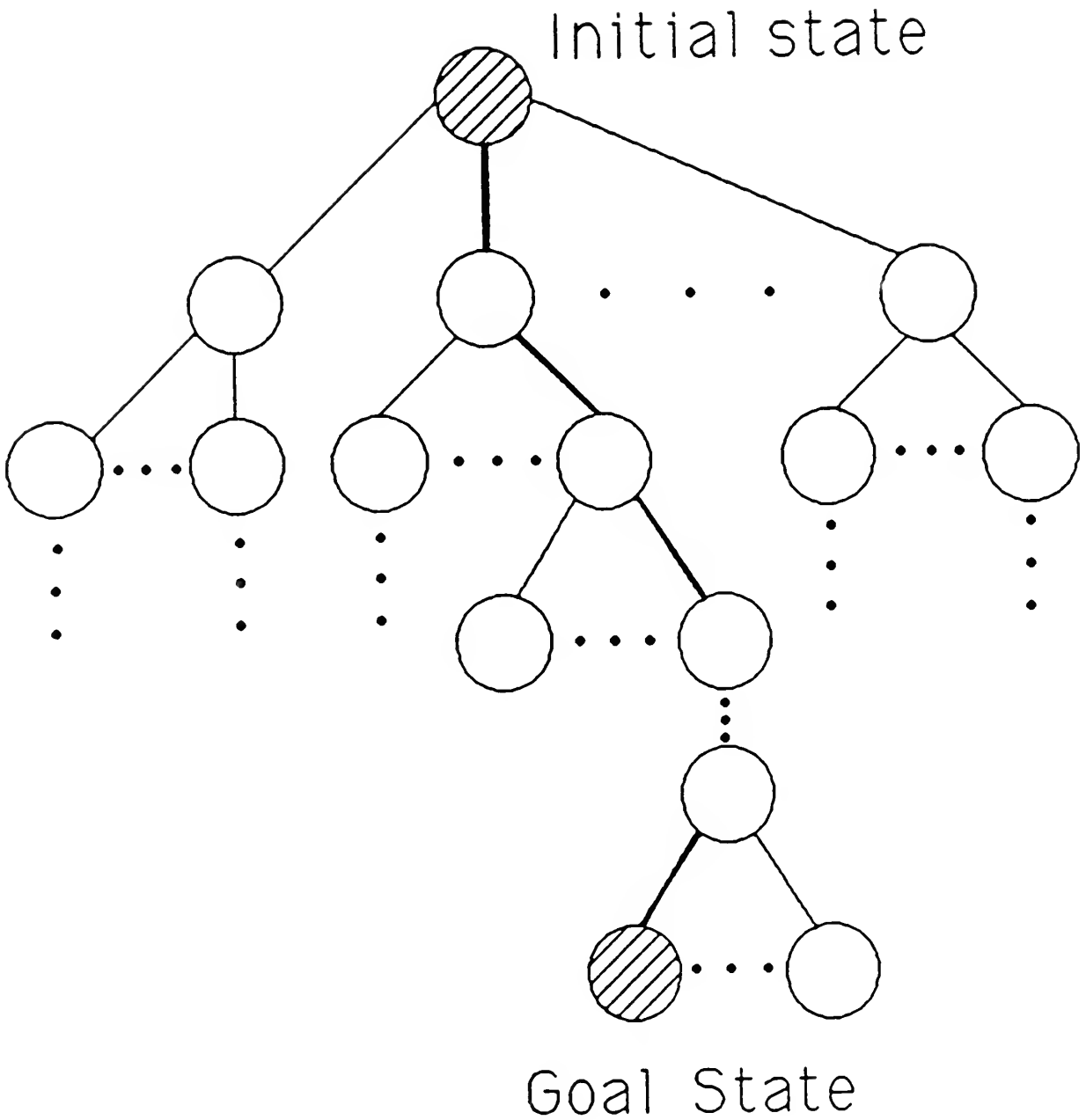


Figure III.1 The Search Tree for Schedule Generation

5) Otherwise expand node n, applying all applicable operators in OP to generate the set of all its successors, T; attach to the nodes in T pointers back to n.

   For every successor x in T, Do

   Begin

6) If x is not already on OPEN or CLOSED, estimate h(x), and calculate

   f(x): = g(x) + h(x) where

   g(x): = g(n) + c(n,x); c(n,x) is the cost of applying the operator corresponding the x.

7) If x is already an OPEN or CLOSED, direct its pointers along the path yielding the lowest g(n).

   End

8) Go to step (2)

End

The foregoing procedure essentially transforms the planning process into a graph search procedure guided by a heuristic function. The procedure keeps two lists in the process of generating the search tree: an OPEN list contains all the nodes yet to be expanded (the leave nodes of the search tree); a CLOSED list contains all the nodes already expanded (the nonleave nodes). For the A* algorithm, the heuristic function f(·) consists of two components: f(n) = g(n) + h(n), where g(n) = the cost of the path found by A* from S_0 to n; and h(n) = the estimated cost of a path from n to S_G.

Insert Figure III.1 Here

The A* algorithm is a best-first searching procedure which uses the heuristic function f(·) to guide the search of the optimal path.

Studies (Nilsson, 1980; Peal, 1983) have shown that when A* employs a perfectly informed heuristics (h=h*) it is propelled directly toward the goal without ever getting sidetracked, spending only M computational steps where M is the number of state-transitions to the goal node. At the other extreme, when no heuristic knowledge is available (c(n,x)=1, h = 0; where c(n,x) represents the cost of applying an operator expanding n to x), the search becomes breadth-first, yielding an exponentially growing complexity.

In generating schedules based on the state-space inference approach, the optimal path consists of a sequence of operators that yield the best schedule. The selection of these operators can be accomplished by algorithm 3.1, where the A* heuristic is used to guide the search. In general, an A* algorithm, such as Algorithm 3.1, guides the search of optimal path by excluding unnecessary node expansions.

Let $C^*_{0,G}$ be the cheapest cost of paths going from S_0 to S_G, i.e., $C^*_{0,G} = h^*(S_0)$. If the cheapest cost of all solution paths constrained to go through n is denoted by f*(n), then

(3.1) $f^*(n) = g^*(n) + h^*(n)$, for any n, and

(3.2) $f^*(n) = C^*_{0,G}$ for $n \in P^*_{S_0,S_G}$

(3.3) $f^*(n) > C^*_{0,G}$ for $n \notin P^*_{S_0,S_G}$

where $P^*_{S_0,S_G}$ stands for the set of optimal path from S_0 to S_G. Then conditions (3.2) and (3.3) imply that following the minimum f* would constitute a perfect search strategy for schedule generation, leading straight toward an optimal solution without ever getting sidetracked.

However, usually information on f*(n) is not possessed, especially the h*(n) component. The A* heuristic is aimed at estimating $C^*_{0,G}$ and providing a measurement on how promising a node is to be on the optimal path. The performance of the A* algorithm is dictated by the following two lemmas:

Lemma 3.1: If there exists a path from the initial state S_0 to the goal state S_G, the A* algorithm must terminate in finite steps.

Lemma 3.2: When the A* algorithm uses a heuristic function that contains an h(.) component such that

(3.4) $0 \leq h(n) \leq h^*(n)$, for every node n,

then the A* algorithm only terminates by finding optimal path to the goal state.

Lemma 3.1 indicates that the A* algorithm is complete, Lemma 3.2 indicates that the A* algorithm is admissible. (A search algorithm that is guaranteed to find an optimal path to a goal, if one exists, is called admissible (Nilsson, 1980)). In Section V, we shall relate the A* heuristic to various scheduling heuristics and compare their performance.

III.2 An Application Example

This section presents an example applying the aforementioned AI scheduling method to the FMS environment. The FMS to be scheduled in this example integrates the operations for flexible machining and assembly.

Suppose the jobs to be done in the system are as shown in Table III.1(a), where jobs 1 and 2 are machining and job 3 is an assembly job. The final part is a cylindrical, as shown in Figure III.1. The machine

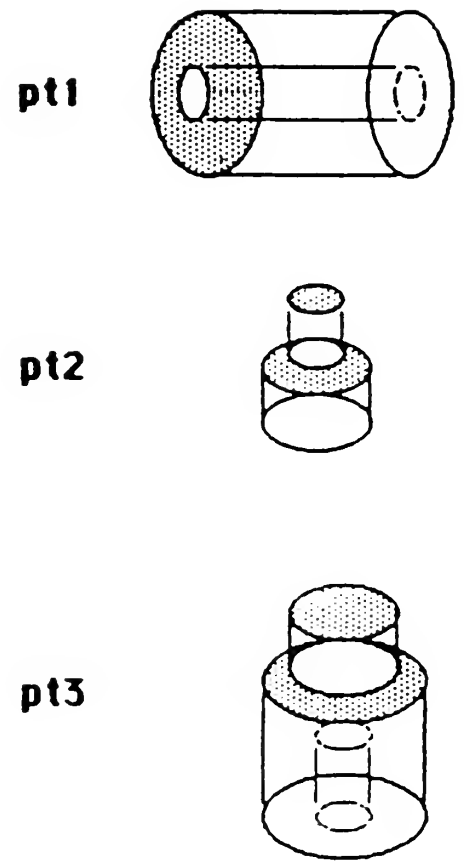


Figure III.2 The Parts Used in the Example

Job	Required Operations	
Job 1 (pt1)	OP4	OP2
Job 2 (pt2)	OP5	OP4
Job 3 (pt3)	pt1	pt2

(a) Job Requirements

Operation	Description
OP2	Drilling
OP4	Cutting
OP5	Turning

(b) Operation Specifications

Machine	Operation	
M2	OP2	OP5
M3	OP4	

(c) Machine Capabilities

Table III.1 Specifications for the Example Problem

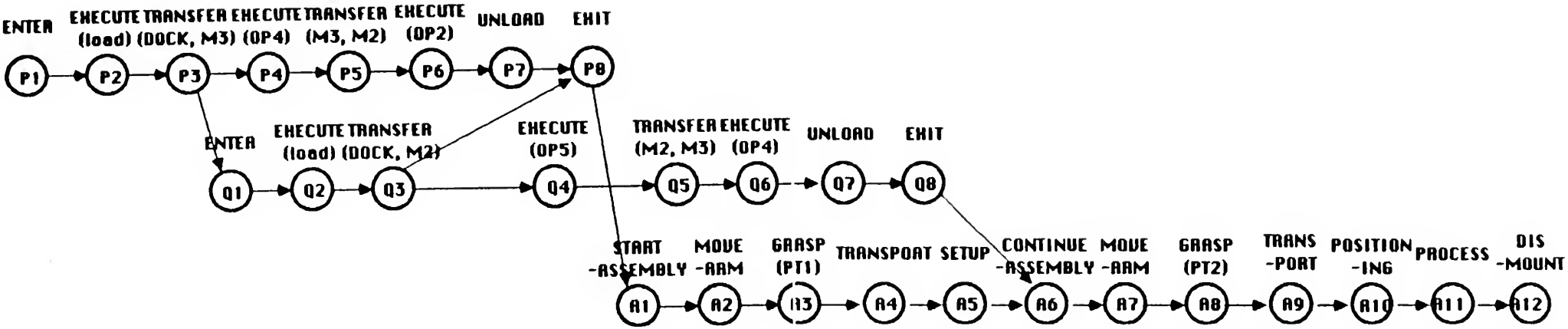


Figure III.3 The Final Schedule

capabilities of the machine are shown in Table III.1(c). A sample set of operators for this example is shown in the Appendix.

The Operator-Search Algorithm is executed to generate a schedule for integrating the machining and assembly operations. The final schedule, a partially ordered network of operators with instantiated operators, is shown in Figure III.3.

Insert Figure III.2, Table III.1, Figure III.3 Here

IV. Nonlinear Planning: A Decomposition Approach

If the FMS scheduling problem is solved by the Operator-Search Algorithm directly, the complexity of the searching procedure can grow prohibitively large (a simpler problem, the job-shop scheduling problem, has been shown to be NP-complete (French, 1982; Gonzalez and Sahni, 1978; Lenstra and Rinnooy Kan, 1978)). In general, the computation complexity of a searching procedure is determined by the number of nodes generated, i.e., by the size of the search tree. For schedule generation, the size of the search tree is bounded by b^d, where b is the branching factor and d is the depth of the tree. For a N-job-M-machine scheduling problem solved by the state-space searching procedure, b is affected by the average number of applicable operators at any node expansion; d is proportional to the average number of operation of each job, N_{op}, i.e., $d = \xi \cdot N_{op}$ where ξ is a constant. For multijob scheduling, both b and d are also proportional to the number of jobs N_J. The size of search tree of Algorithm 3.1 in the worse case would be

$$(b \times N_J)^{\xi} \times N_{op} \times N_J$$

This can grow into a huge tree even for very small-sized problems.

To overcome this complexity difficulty, we employ a decomposition approach. Specifically, a nonlinear planning algorithm is used in the knowledge-based scheduler. It decomposes an n-job-m-machine problem into n subproblems, with each subproblem defined as the routing of one job. Each subproblem is then similar to a single-agent planning problem. Algorithm 3.1 is applied to generate a "plan" for the n subproblems; the primary interactions between these subproblems are their sharing of the machines. The objective of the scheduling problem--to minimize makespan while avoiding conflicting assignments--can be translated into the criterion for the plan-generation problem: to maximize the parallelism while avoiding harmful interactions among the subplans (Sacerdoti (1977), Pednault (1985)). This procedure is shown as follows.

Algorithm 4.1 (Nonlinear Planning)

Begin

(1) Generate a plan for each job by Algorithm 3.1.

(2) Identify conflicting interactions between the planned operators and established precedence constraints to avoid scheduling conflicts.

(3) Use a Plan-Revision Procedure to improve the makespan as much as possible.

End

The Step 2 of this procedure dynamically decides the precedence relationship between two conflicting operators. The underlying principle--based on the least commitment strategy--is not to impose

any precedence constraint unless it is absolutely necessary, so that the parallelism among the subplans is maximized. Information about resources and duration of operators is crucial to the inference engine in making these decisions.

When a precedence constraint is established, the operator restricted by the constraint is put on a list called Alternate-list. In Step 3, the plan-revision step, the scheduling system examines each operator on the Alternate-list and attempts to find if any alternative resource can reduce the queueing time. If such a resource exists, a forward-chaining procedure is used to modify the related section of the plan. This revision procedure is executed for every operator on the Alternate-list, but the forward-chaining procedure is executed only if reassigning the resource can improve the makespan. The Plan-Revision procedure is described as follows.

The Plan-Revision Procedure

(1) Select the first operator on the Alternate-list; identify the resource for which this operator is assigned.

(2) Locate the corresponding critical section of the subplan.

(3) Compare the expected waiting time with the additional processing time by an alternative, idle resource. If no improvement is possible, exit. Otherwise, go to Step 4.

(4) Find out the initial conditions and the ending conditions of this critical section.

(5) Generate a forward-chaining plan that can transform the initial conditions to the goal conditions, using another idle resource.

(6) Modify the subplan by replacing the section identified in Step 2 with the newly generated plan from Step 5.

Based on the performance properties of A* heuristic search, we can derive the following theorem:

Theorem 3.1. The nonlinear planning algorithm is both complete and correct (for proof see Shaw [1986]).

Theorem 3.1 provides two performance guarantees on applying Algorithm 4.1 to the multijob FMS scheduling problem: It can reach the solution in finite steps (completeness) and when it finishes, it always finds the solution if such a solution exists (correctness). However, Algorithm 4.1 is not admissible because of the decomposition--it only achieves suboptimal schedules. The Plan-Revision Procedure is to improve the suboptimal schedule as much as possible by identifying alternative resources for the planned activities.

IV.2 An Example

In this section we shall use an example to illustrate major features of the scheduling approach. This example concerns a 3-machine cell consisting of one CNC lathe and two CNC milling machines; a linear table and two robots are used for material handling, loading/unloading, and transporting workpieces between machines. There is a cell-host supervisory computer in charge of the scheduling and control of the cell. The operation capability (i.e., loading) of each machine is shown in Figure IV.1(a).

Suppose there are three jobs needed to be scheduled, denoted by PT12, PT6, PT16, each job requiring a different set of operations shown in Figure IV.1(b).

Based on algorithm 4.1, the first step is to generate a schedule for each job, resulting in the three schedules shown in Figure IV.2(a). Each schedule, produced by the planning procedure of Algorithm 3.1, consists of a sequence of instantiated operators (only the operator name and the corresponding resource are shown in the figure). The second step of the algorithm is to synthesize the linear schedules into a single schedule by using precedence constraints to resolve conflicts. The final step is to improve the synthesized schedule as much as possible by checking each delayed activity and by employing alternative resources. An instance of plan revision occurs at activities Q3 and Q4 of PT6, which moves from M3 to M2. The final schedule, a partially ordered network, is shown in Figure IV.2(b).

Insert Figures IV.1 and IV.2 Here

IV.3 Dynamic Scheduling

In flexible manufacturing systems, jobs arrive at a manufacturing cell dynamically, each requiring a variety of operations. When new jobs need to be scheduled during the execution of existing jobs, a dynamic version of the nonlinear planning algorithm, such as the following one, can be invoked to accommodate the new jobs.

Step 1. Establish schedules for the new jobs based on the current machines availability shown in the world model;

Step 2. Use the conflict-resolution scheme to coordinate the planned operators for the new jobs and the remaining operators for the old jobs;

Step 3. Improve the modified schedule by the same plan-revision scheme.

The underlying logic of this scheme follows directly from the aforementioned nonlinear planning algorithm. After a tentative schedule is established for the new jobs in Step 1, the forward-chaining procedure is applied to generate a precedence network coordinating the planned operators for the remaining operations of the old jobs and the operators needed by the new jobs. The dynamic scheduling scheme is performed on (1) the unfinished schedule for existing jobs and (2) linear schedules for the new jobs. The same Plan-Revision procedure is then used for achieving minimum makespan. An example of the use of the dynamic scheduling algorithm is shown in Figure IV.3.

Insert Figure IV.3 Here

V. Implementation and Computational Performance

The knowledge-based system for FMS scheduling is written in Common LISP and has been implemented on Explorer, a LISP machine manufactured by Texas Instruments for symbolic processing. The embedded inference engine was a goal-directed, backward-chaining procedure to generate plans and a forward-chaining procedure for plan-revision. To evaluate the performance of the scheduler, we randomly generated ten jobs for testing in a 3-machine cell, where two versions of the non-linear planning algorithm were tested: one with breadth-first search (h(n) = 0); one with A* search (h(n) = estimated remaining processing time). The computation results are shown in Figure V.1. Two interesting observations are worth noting. First, the use of A* heuristic in scheduling significantly reduces the size of the search tree, thus improving the scheduling performance. Second, the numbers of iterations for

RES. \ OP	OP1	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	OP10	OP11	OP12
M1	1	-	-	4	3	7	8	-	-	1	6	5
M2	-	2	-	5	2	-	-	2	3	3	4	4
M3	-	-	3	-	-	6	9	4	1	5	2	6

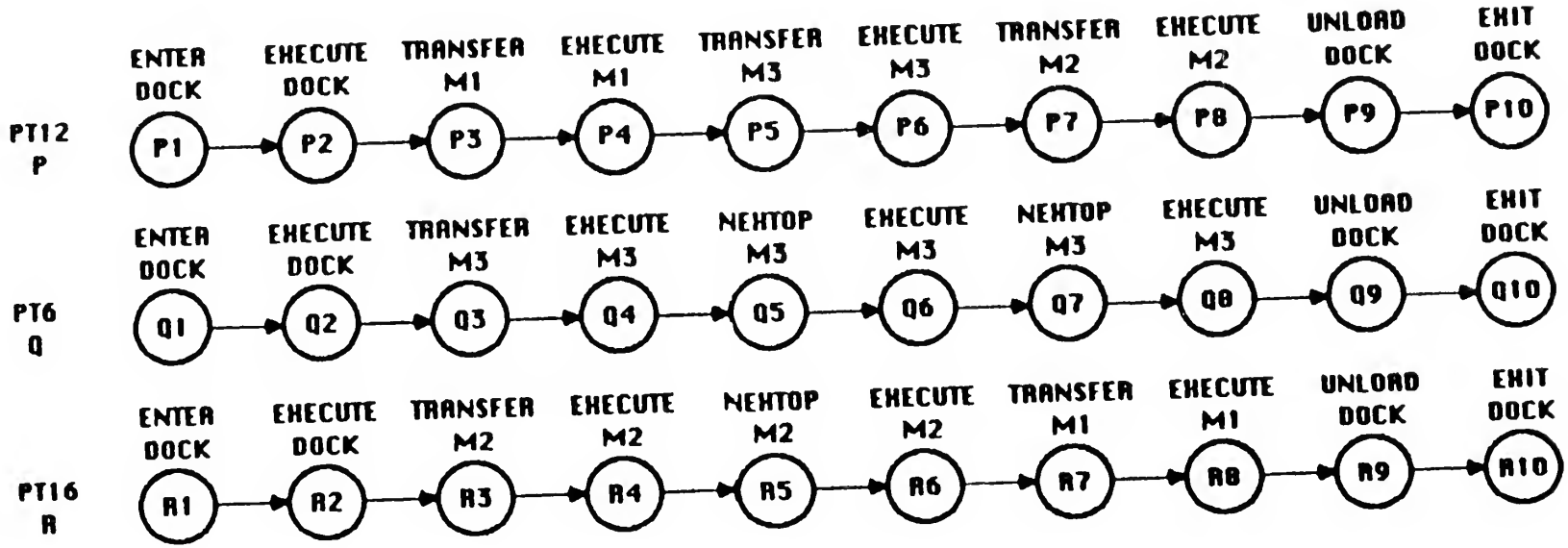
RES. \ OP	LOAD	UN-LOAD
DOCK	5	3

(a)

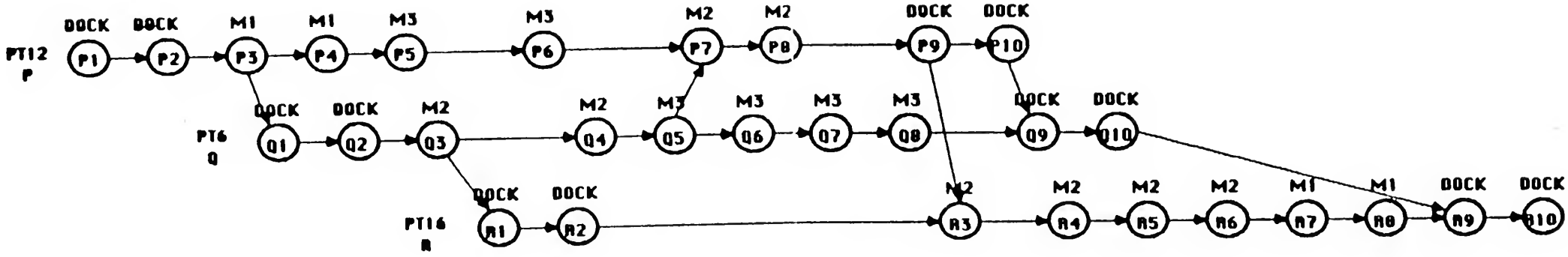
JOB	OP. SEQUENCE			CONTENTION FACTOR
PT6 :	OP9	OP12	OP6	2 3 2
PT12:	OP7	OP3	OP12	2 1 3
PT16:	OP12	OP11	OP10	3 3 3

(b)

Figure IV.1 (a) Operation Loading and Processing Times, (b) Job Specifications

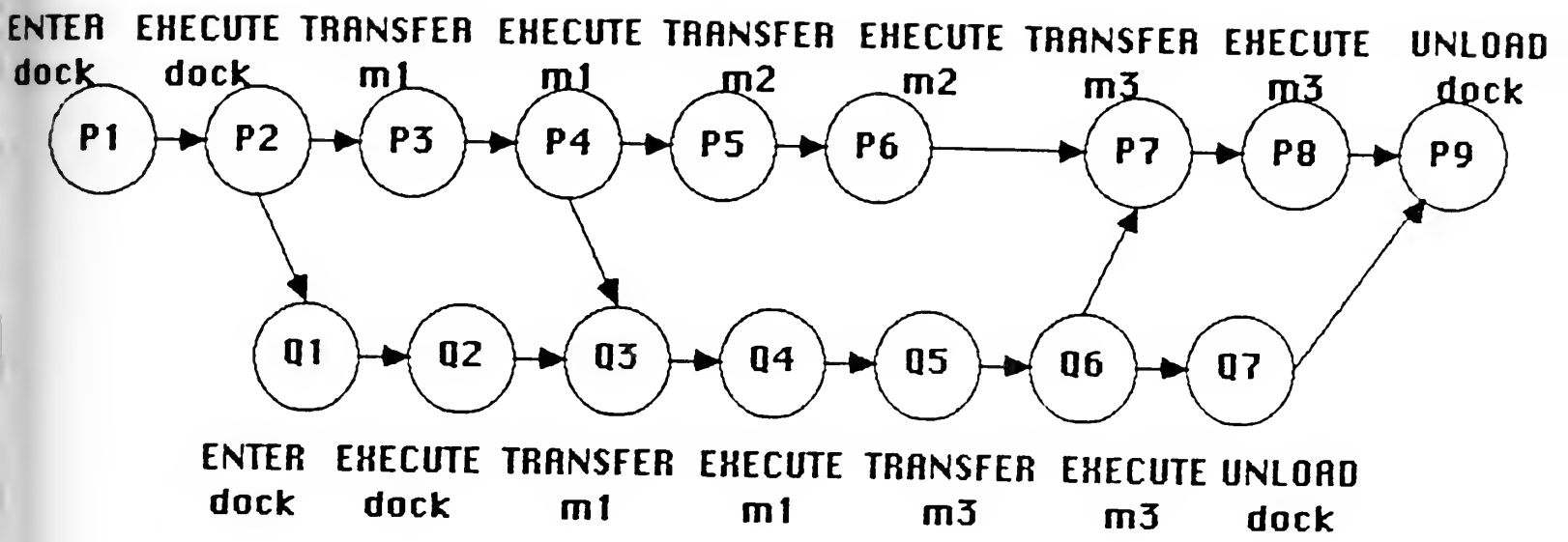


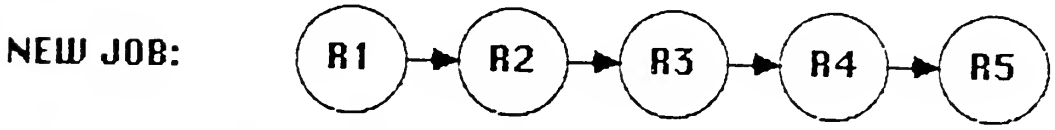
(a)



(b)

Figure IV.2 (a) Linear Schedules, (b) Final Schedules





TRANSFER EXECUTE TRANSFER EXECUTE UNLOAD
m1 m1 m2 m2 dock

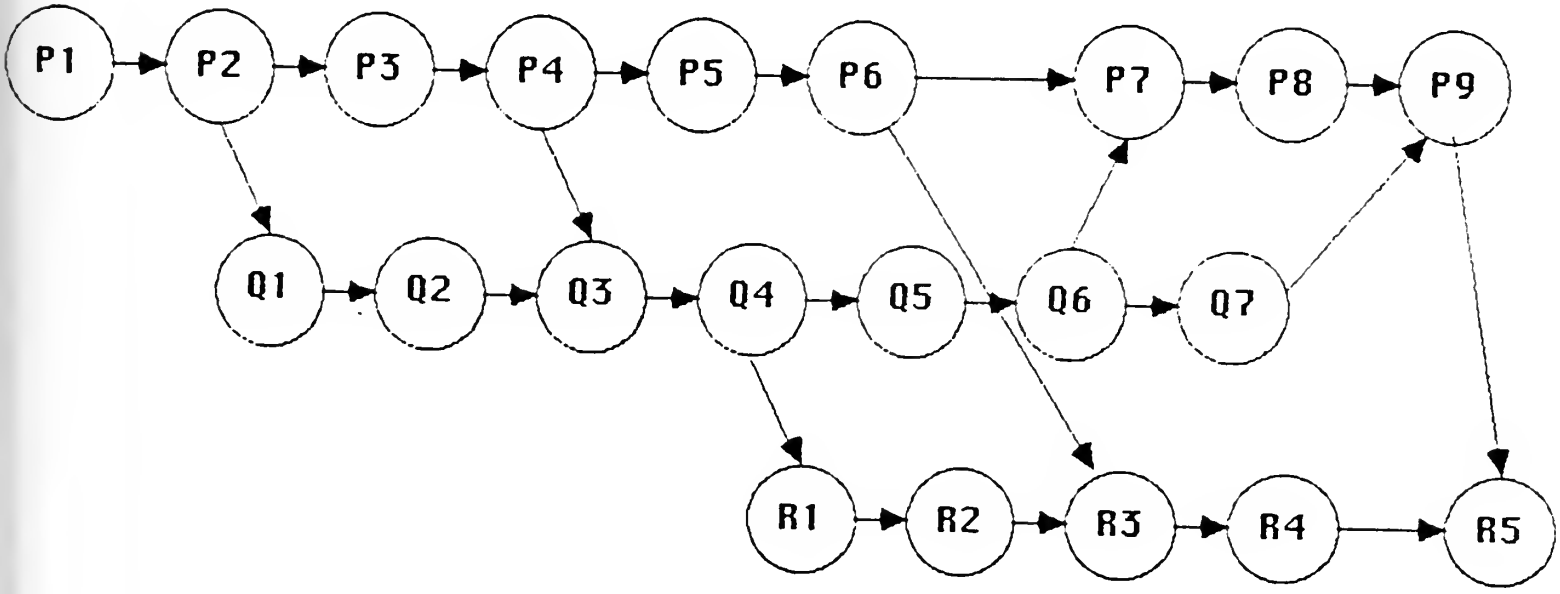


Figure IV.3 An Example of Dynamic Scheduling When a New Job Enters the System

conflict-resolution and plan-revision grow faster than the number of nodes (states) generated. This reflects the decomposition nature of the algorithm.

Insert Figure V.1 Here

In order to compare the impact of heuristic knowledge on scheduling performance, we also conducted a computation experiment to test the scheduling conplexity associated with four different heuristics defined as follows:

$f_0(n) = g(n)$ = height of the search tree

$f_1(n) = g_1(n) + h_1(n)$ = (cumulative processing time) + (estimated total remaining processing time)

$f_2(n) = g_1(n) + h_2(n)$ = (cumulative processing time) + (imminent operation time)

$f_3(n) = g_1(n) + h_3(n)$ = (cumulative processing time) + (number of operations left)

The computation study was conducted on a three-machine FMS cell for which we randomly generated a set of ten jobs, each job requiring three randomly generated operations. The operation loading (i.e., the set of operations assigned to the machines) is also randomly generated from a pool of 12 operations.

The computation results corresponding to the four heuristics are shown in Figure V.2, where the size of the search tree is used to indicate computation complexity. Among the four, $f_1(\cdot)$ results in the best performance, whereas $f_2(\cdot)$ is consistently second-best. This can be explained by the fact that, while both $f_1(\cdot)$ and $f_2(\cdot)$ are admissible

Number of Jobs	Size of the Search Tree	Number of Conflict-Resolution Iterations	Number of Plan-Revision Iterations
2	34	2	1
3	51	5	2
4	66	8	3
5	79	11	5
6	89	15	7
7	112	19	9
8	139	27	11
9	160	30	13
10	183	33	16

Figure V.1 Performance Results of the Knowledge-Based Schedules

(since $h_1(n) \leq h_1^*(n)$ and $h_2(n) \leq h_2^*(n)$), $f_1(\cdot)$ utilizes more global information (Ow (1984)) than $f_2(\cdot)$ does. $f_3(\cdot)$ does not perform as well as $f_1(\cdot)$ or $f_2(\cdot)$ because it does not use processing time information, resulting in a general but weaker heuristic. However, $h_3(\cdot)$ still accounts for some scheduling knowledge by estimating the number of remaining operations, contributing to the better performance of $f_3(\cdot)$ than $f_0(\cdot)$. Accordingly, the computation experiment indicates that (1) a good heuristic knowledge is important in improving the scheduling performance, (2) a global heuristic is better than a local heuristic and (3) a domain-specific heuristic is better than a general heuristic.

Insert Figures V.2 and V.3 Here

The performance of the scheduling system, as measured by the size of the search tree, is also affected by the number of alternatives that have to be enumerated in selecting the solution path. There are two sources for alternative decisions: the number of alternative machines for an operation and the number of applicable activities, as modeled by operators. The former, termed "contention factor," reflects the flexibility of the FMS; the latter, referred to as the "branching factor," indicates the dependency between operators. For example, Figure V.3 contrasts the effect of heuristic f_1 in systems with different levels of contention factors. When the contention factor decreases, the size of the search tree would decrease. Furthermore, the improvement of performance by using good heuristic becomes more significant,

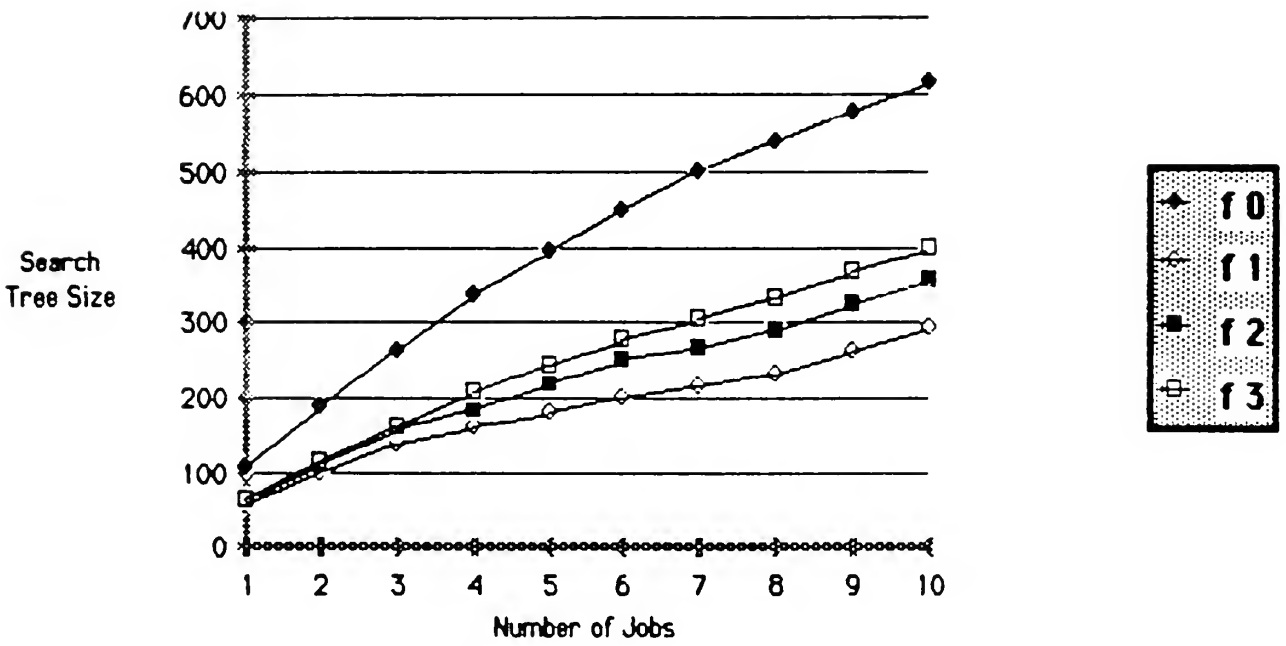


Figure V.2 Performance Comparison of the Four Heuristics

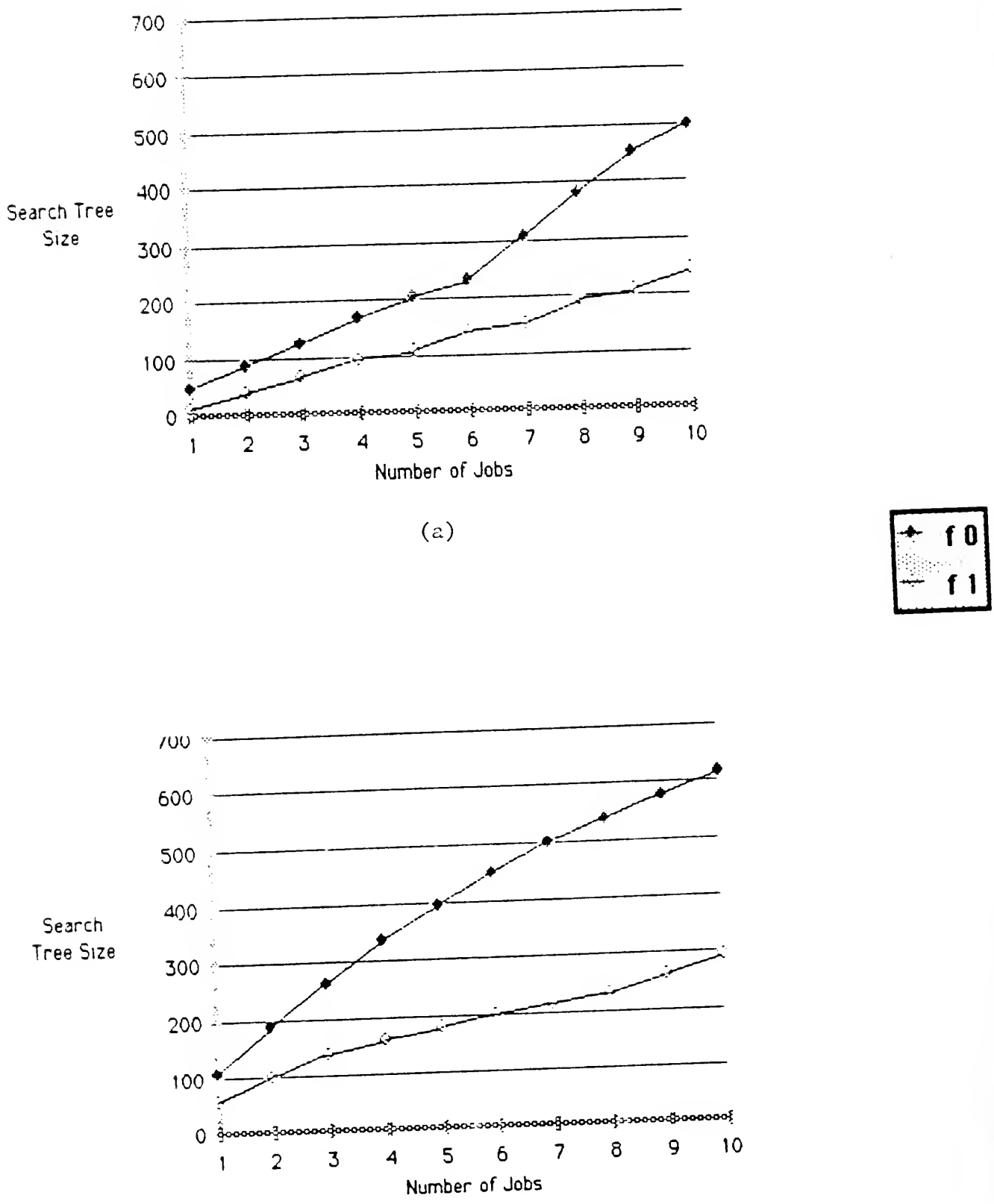


Figure V.3 Scheduling Performance of the Knowledge-Based System in (a) Low Contention and (b) High Contention Cases

as shown in Figures V.3(a) and (b). Thus, using a good heuristic becomes even more important in the FMS when the degree of flexibility increases and machines are set up for a wide variety of operations.

VI. Conclusion

This paper describes a knowledge-based scheduler for FMSs. Organized as a hierarchical system, the scheduler can perform scheduling/rescheduling in order to handle the dynamically changing FMS environment. By using a symbolic world model and the automatic planning technique for generating schedules, the knowledge-based scheduler can schedule jobs in an FMS in real time and assign resources dynamically; these features are important for FMS scheduling because of the shorter lead-times and versatile machines. We have also shown the importance of heuristic knowledge in expediting the scheduling process. A decomposition algorithm, similar to nonlinear planning, is employed for schedule-generation with reduced complexity.

Acknowledgements

This research is supported in part by grants from the Office of Information Management sponsored by IBM, the Research Board of the University of Illinois, and the Amoco Foundation Professorship. Thanks are due to S. C. Park for his assistance in the computation study and to Texas Instruments for the supports they have given to the AI/Expert System Programming Lab.

Appendix Sample Operators in the Knowledge Base

TRANSFER (?m ?mp ?op ?opp ?pt)
Precondition: finish-op (?m ?op ?pt) & pt-nextop (?op ?opp ?pt)
& mach-op (?mp ?opp) & different (?m ?mp)
Add-list: mach-pt (?mp ?opp ?pt) & idle (?m)
Delete-list: finish-op (?m ?op ?pt) & pt-nextop (?op ?opp ?pt)
Resource: ?mp
Duration: 2

NEXTOP (?m ?op ?opp ?pt)
Precondition: finish-op (?m ?op ?pt) & pt-nextop (?op ?opp ?pt)
& mach-op (?m ?opp)
Add-list: mach-pt (?m ?opp ?pt)
Delete-list: finish-op (?m ?op ?pt) & pt-nextop (?op ?opp ?pt)
Resource: ?m
Duration: 0

UNLOAD (?m dock ?op ?pt)
Precondition: finish-op (?m ?op ?pt) & pt-nextop (?op nil ?pt)
Add-list: mach-pt (dock un-load ?pt) & idle (?m)
Delete-list: finish-op (?m ?op ?pt) & pt-nextop (?op nil ?pt)
Resource: dock
Duration: 3

EXIT (?pt)
Precondition: mach-pt (dock un-load ?pt) & idle (dock)
Add-list: done (?pt) & in-buffer (?pt)
Delete-list: mach-pt (dock un-load ?pt)
Resource: dock
Duration: 1

SETUP (?subpt ?pt arm)
Precondition: first-subpt (?subpt ?pt) & prepared (?subpt ?pt)
Add-list: finish-subpt (?subpt ?pt) & idle (arm)
Delete-list: first-subpt (?subpt ?pt) & prepared (?subpt ?pt)
& in-arm (?subpt ?pt)
Resource: arm
Duration: 1

MOVE-ARM (arm ?p ?subpt ?pt buffer)
Precondition: idle (arm) & position (arm ?p)
& ready-to-assemble (?subpt ?pt)
Add-list: position (arm buffer) & ready-to-grasp (?subpt ?pt)
Delete-list: idle (arm) & position (arm ?p)
& ready-to-assemble (?subpt ?pt)
Resource: arm
Duration: 2

GRASP (?subpt arm buffer ?pt)
Precondition: position (arm buffer) & ready-to-grasp (?subpt ?pt)
Add-list: in-arm (?subpt ?pt)
Delete-list: in-buffer (?subpt) & ready-to-grasp (?subpt ?pt)
Resource: arm
Duration: 1

EXECUTE (?m ?op ?pt)
Precondition: mach-pt (?m ?op ?pt) & idle (?m) & mach-op (?m ?op)
Add-list: finish-op (?m ?op ?pt)
Delete-list: mach-pt (?m ?op ?pt) & idle (?m)
Resource: ?m
Duration: (lookup (?m ?op))

References

1. Brund, G., Elia A., and Laface, P., 1986, "A Rule-Based System to Schedule Production," IEEE Computer, Vol. 19, No. 7, p. 32-40.

2. Buzacott, J. A. and Yao, D., 1986, "Flexible Manufacturing Systems: Review of Analytical Models," Management Science, Vol. 32, No. 7, p. 890-905.

3. Cutkosky, et al., 1983, "Precision Machining Cells within a Manufacturing System," Robotics Institute, Carnegie-Mellon University.

4. Fikes, R. E. and Nilsson, N. J., 1971, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," Artificial Intelligence, Vol. 2 (3/4), p. 189-208.

5. Fikes, R. E., Hart, P. E. and Nilsson, N. J., 1972, Learning and Executing Generalized Robot Plans," Artificial Intelligence, 3(4), p. 251-288.

6. Fox, M. S., 1983, "Constraint-Directed Search: A Case Study of Job-Shop Scheduling," Ph.D. Thesis, Tech. Report CMU-RI-TR-83-22, Robotics Institute, Carnegie-Mellon University.

7. French, S., 1982, Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop, John Wiley, New York.

8. Gershwin, S. B., Akella, R., and Choong, Y. F., 1985, "Short-term Production Scheduling of an Automated Manufacturing Facility," IBM J. of Research and Development Vol. 29, No. 4, p. 392-400.

9. Gonzalez, T. and Sahni, S., 1978, "Flowshop and Jobshop Schedules: Complexity and Approximation," Operations Research, 26, p. 36-52.

10. Grant, T., 1986, "Lessons for O.R. from A.I.: A Scheduling Case Study," J. Op. Res. Soc., Vol. 37, No. 1, p. 41-57.

11. Hayes-Roth, F. and Waterman, D., 1978, "Principles of Pattern-Directed Inference Systems," in Pattern-Directed Inference Systems, Waterman, D. and Hayes-Roth, F. (Eds.), Academic Press, New York.

12. Hitz, K. L., 1979, "Scheduling of Flexible Flow Shops," MIT Laboratory for Information and Decision System Report LIDS-R879.

13. Jones, A. T. and McLean, C. R., 1986, "A Proposed Hierarchical Control Model for Automated Manufacturing Systems," Journal of Manufacturing Systems, Vol. 5, No. 1, p. 15-25.

14. Kimemia, J. and S. B. Gershwin, 1985, "Flow Optimization in Flexible Manufacturing Systems," International Journal of Production Research, Vol. 23, No. 1, 81-96.

15. Kusiak, A., 1986a, "Scheduling Flexible Manufacturing and Assembly Systems," in Proc. of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems, K. Stecke and R. Suri (Eds.), Elsevier Science Publishers, Amsterdam.

16. Kusiak, A., 1986b, "Artificial Intelligence and Operations Research in Flexible Manufacturing Systems," Working Paper #10186, Dept. of Mechanical and Industrial Engineering (University of Manitoba: Winnipeg, Manitoba).

17. Lenstra, J. K. and A. H. G. Rinnooy Kan, 1978, "Complexity of Scheduling under Precedence Constraints," Operations Research, Vol. 26, No. 1, 22-35.

18. Merchang, M. E., 1983, "Production: A Dynamic Challenge," IEEE Spectrum, p. 36-39.

19. Newell, A. and Simon, H., 1972, Human Problem Solving, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

20. Nilsson, N., 1980, Principles of Artificial Intelligence Tiago, Palo Alto, CA.

21. Nof, S., Barash, M., and Solberg, J., 1979, "Operational Control of Item Flow in Versatile Manufacturing Systems," International J. of Production Research, Vol. 17, No. 5, p. 479-489.

22. Ow, P. S., 1984, "Heuristic Knowledge and Search for Scheduling," unpublished Ph.D. thesis, Graduate School of Industrial Administration, The Robotics Institute, Carnigie-Mellon University, Pittsburgh, PA.

23. Park, S. and Shaw, M. J., 1986, "Incorporating Sequencing Heuristics in an AI-based FMS Scheduling System," Technical Report, Department of Business Administration, University of Illinois, Champaign, IL.

24. Pearl, J., 1983, "Knowledge vs. search: a quantitative analysis using A*," Artificial Intelligence, Vol. 20, p. 1-13.

25. Pednault, E., 1985, "Preliminary Report on a Theory of Plan Synthesis," Technical Note 358, SRI International, Menlo Park, CA.

26. Ranky, P., 1983, The Design and Operation of Flexible Manufacturing Systems, IFS Publications, North-Holland.

27. Sacerdoti, E. D., 1977, A Structure for Plans and Behavior North-Holland, New York.

28. Shaw, M. J., 1986, "A Pattern-Directed Approach to FMS Scheduling," in Proc. of the Second ORSA/TIMS Conference on Flexible Manufacturing Systems, K. Stecke and R. Suri (Eds.), Elsevier Science

Publishers, Amsterdam. A longer version appeared in Working Paper No. 1301, Department of Business Administration, University of Illinois.

29. Shaw, M. J. and Whinston, A. B., 1985a "Automatic planning and Flexible Scheduling: A Knowledge-Based Approach," Proceedings of International Conference on Automation and Robotics, St. Louis. Also appeared in Robotics and Industrial Engineering: Selected Readings, Vol. II, Fisher, E. and Maimon, O. (Eds.), Industrial Engineering and Management Press.

30. Shaw, M. J. and Whinston, A. B., 1985b, "Task Bidding, Distributed Planning, and Flexible Manufacturing," Proceedings of IEEE Conference on Artificial Intelligence Applications, Miami, FL.

31. Shaw, M. J. and Whinston, A. B., 1986c, "Application of Artificial Intelligence Techniques to Planning and Scheduling," Flexible Manufacturing Systems: Methods and Studies, A. Kusiak (Ed.) Amsterdam, North Holland.

32. Stecke, K. E., 1983, "Formulation and Solution of Nonlinear Integer Production Planning Problems for Flexible Manufacturing Systems," Management Science, Vol. 29, No. 3, March, p. 273-288.

33. Vere, S., 1983, "Planning in Time: Windows and Durations for Activities and Goals," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, No. 3, p. 246-266.

34. Wilkins, D., 1984, "Domain Independent Planning: Representation and Plan Generation," Artificial Intelligence, Vol. 22, p. 269-301.